Implementation of collision avoidance algorithm on real-time IoT cloud platform for swarm robotics

**Abstract:** The IoT Cloud is a cloud platform for implementing applications that remotely control smart devices or process huge amount of real time stream data from massive physical devices. Such platform shifts computation load from the device to the cloud, which provides more powerful process ability to a simple device. In Swarm robotics, robots are supposed to be small, energy efficient and low-cost, but still be smart enough to carry out individual and swarm intelligence. These two goals are normally contradictory to each other. Besides, in real world robot control, real time on-line data processing is required, but most of the current Cloud Robotic Systems are focusing on off-line batch processing. However, the IoT Cloud may provide a way that leads this research area out of this dilemma. This paper explores the availability of IoT Cloud for real time control of massive complex robots by implementing a relatively complicated but better performed local collision avoidance algorithm on the platform. The IoT Cloud application and also the IoT Cloud Driver, which connects the robot and the Cloud, are developed and are all deployed in the IoT Cloud. Simulation tests are carried out and the result shows that, when the number of robots increases, by simply scaling the computation resources for the application, the algorithm can still maintains a good performance. Such characteristics provide a new approach for studying massive complex robots in swarm robotics.

**Key word:** Internet of things, cloud computing, swarm robotics, swarm intelligence, collision avoidance, real time stream processing

**I. INTRODUCTION**

Since Cloud computing can provide elastic, on demand, ubiquitous worldwide accessible computing and storage resources, it has been introduced into various areas from big data analysis to real time robot control. One very promising area is developing universal platform for Internet of Things(IoT)[[1](#_ENREF_1)] applications using cloud computing technology. The IoT Cloud system is normally featured as both real time responding and big data processing.  As large number of smart devices are connected to the IoT Cloud, massive real time stream data from these devices needs to be handled before it can be recorded in the database and processed off line by cloud. In some scenarios, such as robot control, the stream data from devices has to be processed and feed back in time. These time-critical tasks require the system to respond fast enough, thus cloud computing techniques such as MapReduce is not viable for this kind of applications.

However parallel processing ability and cluster computing framework of techniques like MapReduce are very appealing, especially to systems that need to deal with large number of computation intensive entities. Swarm robotics[[2](#_ENREF_2)] is a typical research area that commonly deals with such systems. In swarm robotics, normally, the robot should be as small and energy efficient as possible[[3](#_ENREF_3), [4](#_ENREF_4)], but still it needs to be able to perform basic behaviors of the intelligent entity under research and can also carry out high level swarm intelligence[[5](#_ENREF_5)]. For traditional robotic system, these two aspects are contradictory to each other. But with the help of cloud computing, most of the computation can be offloaded to the cloud and, by utilizing elastic cloud computing, the number of robots in a swarm can scale flexibly. As most of the computation is transferred into the cloud, the on-board system of a robot can be greatly reduced, keeping only very simple sensation and actuation modules and leaving all high level algorithms to the cloud. Motivated by such demand, several cloud platforms[[6-10](#_ENREF_6)] dedicated to robotic control are designed. Nevertheless, most of these systems mainly focus on static data processing, such as object recognition, path planning and so on. These tasks are not so time-critical as those dynamic tasks such as local collision avoidance[[11](#_ENREF_11)].

The IoT cloud platform, developed on real time distributed processing framework, is a scalable real time stream data processing system[[1](#_ENREF_1)]. This platform is much more suitable for time-critical applications as it processes stream data in real time response. The core of the IoT cloud is a distributed real time stream computing engine. Data from devices or databases can be injected into the engine as streams and the computing logic running in the engine will process the data and then emit results. As long as the processing logic is running, data streams can be processed in time. The computing engines utilizes cluster computing paradigm, which makes it easily scalable and fault-tolerant.

This paper explores the parallelism and scalability of the IoT cloud platform in real time data processing by implementing multi-robot collision avoidance. Unlike other algorithm parallelism research, this paper focuses on entity or agent level parallelism and studies mainly on computation resource scaling according to robot number. And also, unlike normal swarm robotic researches that seek for simplified model to reduce computation, this paper implements more complicated algorithm that can reflect more details about the physical system and can be used in real world scenario. The results of the experiment demonstrate that the IoT cloud introduced in this paper is an effective, scalable platform for swarm robotics. The main contribution of this paper is exploring novel cloud framework for implementation of computation intensive algorithms in swarm robotics.

The remainder of this paper is organized as follows: Section II briefly introduces collision avoidance theory and related algorithms. Section III describes the architecture of thIoT Cloud Platform. Section IV explains the design of the cloud application in detail. Section V presents experiment of the whole system and the application, and analyzes the results. In the end, Section VI summarize and conclude the whole paper.

**II. LOCAL COLLISION AVOIDANCE FOR NON-HOLONOMIC ROBOTS**

Local collision avoidance is one of the most important aspect in robot navigation. The task of local collision avoidance is to dynamically compute the optimal collision free velocity for a robot, which is based on the observation of the environment. Unlike motion and path planning that have static knowledge of the global environment and make one-time plan, local collision avoidance needs to respond to the dynamics of the environment[[11](#_ENREF_11)] such as other active entities or obstacles that are not reflected in the static map.

Current local collision avoidance methods are mainly based on the Velocity Obstacle(VO) theory[[12](#_ENREF_12)]. VOs are areas in velocity space that if the velocity of a robot points into one of the area it will collide with another robot after some time. A diagram of VO is shown in Fig. 1. Several types of VO are defined according to the different VO calculation methods. The Reciprocal Velocity Obstacle(RVO)[[13](#_ENREF_13)] split collision avoidance responsibility equally between the two robots that may collide with each other, while the Hybrid RVO(HRVO)[[14](#_ENREF_14)] translates apex of RVO to the intersection of the closest RVO leg to own velocity and the leg of VO furthest from own velocity, which encourages choosing of preferred side and reduces the chance of a reciprocal dance.

All these VO, RVO and HRVO assumes that the robot can reach any velocity in the velocity space, one hundred percent accurate localization information and circular robot footprint. However, the real robot cannot satisfy such prerequisites. Therefore other constraints need to be attached to those VOs. These constraints include kinetic constraint such as acceleration and max velocity limits, Non-Holonomicconstraints[[15](#_ENREF_15)] for differential robot, and localization uncertainty[[16](#_ENREF_16)]. And when considering localization uncertainty, robot footprint needs to be expanded. Simply use circular footprint with extended radius may exclude possible valid velocities. So convex hull footprint[[17](#_ENREF_17)] calculated from Minkowski Sum of robots and obstacles is introduced in calculating VOs. The calculation of convex hull footprint for a robot is highly computation intensive and it may take around 50 percent of the overall algorithm computation time.

Fig. 1 Velocity obstacle introduced by robot B

Once all VOs are obtained from velocity space, optimization algorithm need to be designed to select optimal velocity from the areas outside all VOs. There are three main methods for collision free velocity selection which are Optimal Reciprocal Collision Avoidance(ORCA) method, Clear Path method[[18](#_ENREF_18)] and Sampling based method. According to [[17](#_ENREF_17)], Clear Path method has relatively better overall performance in real world experiments.

With all the detailed considerations above, an algorithm that can control robots in real world to avoid each other in a more effective way is developed. But such algorithm requires at least a laptop to run. In swarm robotics the number of robots can reach about one hundred or more, and, obviously, equipping a laptop for each robot can greatly increase investment and also the size of the robot. Besides power consumption of a laptop will lead to less robot running time.

To use the algorithm but at the same time reduce the “side effects”, one effective approach is offloading algorithm computation into cloud environment and connecting the robot through wireless network[[19](#_ENREF_19)]. In the following sections details about the cloud platform and implementation of the algorithm will be illustrated. In this paper, the algorithm, which uses convex hull footprint for VO calculating, considers all aforementioned constraints, and utilize Clear Path method for optimal velocity computation, is implemented.

**III. IOT CLOUD ARCHITECTURE**

The IoTCloud[[1](#_ENREF_1)] is a platform that provides cloud services for vast number of Internet accessible devices. The IoT Cloud mainly consists of three layers: Front-end Gateway Layer, Stream Processing Middle Layer and Batch/Storage Back-end Layer. The three layers are connected via message broker and coordinated by Zookeeper. Fig. 2depicts all major components of the system.

Fig. 2 The architecture of IoT Cloud

The Front-end Gateway Layer is responsible for connecting devices with the Message broker. As IoT Cloud is designed to serve heterogeneous devices, it needs a component to maintain specific information about the devices and map between message broker channels and native device data channels. All devices are connected through Gateways, and below the Gateways are IoT Cloud device drivers which convert device data into message that broker can process. Such device drivers first get data from devices and then call IoT Cloud APIs to send converted data into the cloud. The Gateway maintains connections between devices and the cloud, while there is a Gateway master that coordinates multiple Gateways and registers connection information, such as channel maps between message broker and devices, so that the Middle layer can discover devices and provide service entries.

The Stream Processing Middle Layer handles real-time stream data processing. This layer uses Apache Storm[[20](#_ENREF_20)] as the computation engine. Storm is a distributed real-time streaming processing system that can process utmost one million of Tuples(data type processed in Storm) per second. It is very suitable to process stream data from large number of smart devices. Storm gets source data from its component called Spout and then sends data to the process component called Bolt. Spouts and Bolts are organized into process logic called Topology. To use the IoT Cloud service, application topology should be developed first. As in IoT Cloud, data input and output of the application topology are closely related to devices, so the IoT Cloud platform provide APIs that can be used to build custom input Spouts and output Bolts. As mentioned before the Gateway layer is responsible for maintaining the connections of Spouts and Bolts to the message broker by writing connection information in Zookeeper[[21](#_ENREF_21)]. To use the real time stream processing service in this layer, data from devices should be sent to the correct message broker channels, which are connected to the input Spouts of an application, via IoT Cloud device drivers and then, by subscribing the channels that connect to the output Bolts of the application topology, results can be fetched in time. Such data processing paradigm is just what robotic controlling needs. Most of the work in this paper focuses on designing and implementation of application topology in this layer and corresponding IoT Cloud driver for robot collision avoidance. And most of the computation required for the collision avoidance algorithm is shifted to this layer. Once an application is deployed into the IoT Cloud, it can provide services to large number of devices as long as they have the correct IoT Cloud drivers. By deploying multiple instances of the application or increasing number of computation nodes for the application, data processing ability can be scaled accordingly.

The Batch/Storage Layer stores data from Stream Processing Middle Layer and provide batch processing/data mining services for the static data from various distributed databases. Since this paper mainly works on real time data processing and controlling, this layer will not be used.

**IV. IMPLEMENTATION OF THE COLLISION AVOIDANCE ALGORITHM**

*A. Application overview*

Fig. 3 shows the overall design of the collision avoidance application. On the front-end Gateway layer there is IoT Cloud driver module which communicates with device and converts data between message broker and local device. The driver is deployed on the Gateway site of IoT Cloud, which runs on a local desktop machine, and managed by the Gateway. As most robots support Robot Operation System(ROS)[[22](#_ENREF_22)], here ROS is adopted as the device driver that interacts directly with the robots. The IoT Cloud driver will subscribe ROS topics to get robot state, such as odometry, laser scans and so on, and then convert the data into message that can be transmitted through broker to the cloud. After finishing processing the data, the IoT Cloud will send back velocity command through message broker and the IoT Cloud driver will convert the message into ROS message type, and then send the ROS command message to the correct topic so that the robot can be controlled. This paper uses ROS as device driver just for demonstration, however besides ROS, other device specific driver can also be used, as long as they can provide APIs for data retrieving.

Fig. 3 Overview of the Collision Avoidance application

The IoT Cloud computation engine together with the message broker servers are deployed in the FutureGrid[[1](#_ENREF_1)] cloud platform. The complicated collision avoidance algorithm is implemented as a Storm Topology and running in the computation engine. The message broker of the IoT Cloud relays data from IoTcloud driver and feeds it into Spouts of the control topology. When the velocity command is calculated, the topology will send the command back through an output Bolt to message broker which then relays the message back to the cloud driver, and then to the robot.

*B. IoT Cloud driver for collision avoidance*

Iot Cloud driver is used to connect devices to the IoT Cloud Platform. For different types of devices, the Cloud driver are different but for the same type of devices they can use the same Cloud driver and just need to spawn a new instance for each device. To perform collision avoidance, odometry, laser scan and pose array of the robot need to be sent to the cloud. These informations are published by the robot through ROS topics as shown in Fig. 4. So the IoT Cloud driver first subscribes these topics and get the ROS messages. However these ROS messages are not viable for message brokers, such as Rabbitmq which is used in this work, and it is the cloud driver that convert these messages into custom defined data types that can be serialized/deserialized and sent by the message broker.

Fig. 4 IoT Cloud driver for collision avoidance

The next thing that an IoT Cloud driver needs to do is to define IoT Cloud Channels for those robot information. As for IoT Cloud application topology, each input Spout or output Bolt is connected to a predefined IoT Cloud Channel according to the application and the message broker. All data is transmitted through these Channels. As the number of Spout and Bolt in an application topology cannot be changed, but the number of the robot that connects to the cloud may vary from time to time, so IoT Cloud Channels should be defined according to the robot information types rather than the robot entity. So the IoT Cloud driver will create an IoT Cloud Channel for each information type and publish converted message to the corresponding Cloud Channel. This paradigm of publishing messages from different cloud driver instances into one IoT Cloud Channel is called Grouping in the IoT Cloud platform. To distinguish the messages from different robots, the unique robot ID generated by cloud driver is attached to the message. However, the Bolt of the application topology that publishes velocity command back to the robot will also publish all commands for different robots into one IoT Cloud Channel. In this case, to ensure the cloud driver instance receive its own velocity command messages, the Grouping flag for that Channel should be set to False. Then the Gateway will create a queue for each cloud driver instance and messages from that Channel will automatically be sent to the correct queue according to the robot ID attached.

*C. Topology design*

All the algorithm and control logic for IoT Cloud based collision avoidance are implemented in the Storm Topology. Before design the application topology, some details of the collision avoidance application that implemented in local mode should be explored. The collision avoidance algorithm introduced in section II can be summarized as Fig. 5.

Fig. 5 The collision avoidance algorithm

The whole algorithm runs as a control loop which executes periodically. For a single loop, it starts from collecting robot newest state information, including getting obstacles from laser scan, calculating convex hull footprint from pose array, getting neighbors from pose share messages and extract velocity and pose of the robot from odometry. The next step is to calculate preferred velocity from global plan. This paper just implements a very simple global planner that generates a straight path consisting of a number of way points from the start to the goal. If the robot has already reached the goal position and only needs to adjust its heading, then stopping velocity command or rotating command will be sent to the robot directly. Otherwise, algorithm will update the robot and its neighbors’ position according to the information that each robot shares. With all the information up to date, VO lines from different aspects will be calculated. Such VO lines includes those from neighbors, obstacles and various constraints, such as aforementioned kinetic constraints, non-holonomic constraints and so on. Once all VO lines are obtained, the optimal velocity that is closest to preferred velocity is selected using Clear Path algorithm. Finally the application will check the validation of the computed new velocity. If it is valid, then the velocity will be sent to the robot, otherwise the application will try next way point and calculate new velocity.

To implement the whole application into a Storm Topology, Spouts and Bolts that connect the topology and the message broker should be design first. As there are five types of information that needs to be input into the Topology, five Spouts need to be defined. These Spouts include odometry receiver Spout, scan receiver Spout, pose array receiver Spout, Configuration Spout and pose share receiver Spout. The first three Spouts are used to get robot state information, while the Configuration Spout receives basic parameters of the robot, such as control frequency, acceleration limits, maximum velocity, start pose and goal pose and so on, and the pose share receiver spout is responsible for receiving information about all robot neighbors. Two Bolts are required for publishing the computed velocity command and pose share messages to the message broker. As the algorithm need neighbors' information, all robots in the scene should publish their state to a common IoT Cloud Channel periodically, so that they can share their newest state with each other. Both of these Spouts and Bolts are defined in a configuration file and the IoT Cloud platform will automatically generate them according to this file.

The rest of the components of the application in Fig. 5 can be implemented in different ways. For example, all of the rest components can be integrated into a single one Bolt or, on the contrary, each of the component can be implemented into a Bolt. Three possible Topologies are designed as shown in Fig. 6.

Fig. 6 Three topologies for collision avoidanceapplication

Topology A integrates all components into one Bolt. As all messages are feed into the Bolt, the Bolt is too busy dealing with new messages that the overall delay of the velocity command is very high. Topology C implements each component into separate Bolt and even calculate different types of VO lines in parallel. However robot state information has to be transmitted through too many Bolts, the serialization/deserialization process along with the communication delay between computation nodes consumes much more time than the time that is saved by parallel computing. So the overall delay of Topology C is also very high. By reviewing the performance metric of Topology A and C, it shows that Bolts that process input messages from Spouts take much less computation than the Bolts that calculate VO lines and velocity commands. So in Topology B, all components that process robot state information are combined into one Bolt and other components that calculate VO lines and velocities are wrapped into another Bolt. Such design can reduce delays caused by data transmission between Bolts and, at the same time, isolate message processing from the main collision avoidance algorithm.

Besides those Spouts and Bolts that interact with message broker, there are five more components in Topology B. To utilize collision avoidance control service, a robot should first send its parameters and start and goal poses to the Global Planner Bolt through Configuration Spout. The Global Planner Bolt will then do the following jobs:

* Make a global path plan according to the start and goal poses.
* Spawn an Agent object that contains robot parameters, some algorithm related state variables, and the global plan generated before, and send it to Velocity Compute Bolt.
* Spawn a Control-Publish Time State object that contains control period and pose share period, and also two variables that record last time that the robot is controlled/published respectively. Besides, there are too boolean variables that record whether, currently, the Topology is calculating velocity or publishing pose share message. This object will be sent to the Dispatcher Bolt that triggers control or pose share process according to the given period.
* Spawn a Pose Share Message object that contains basic information to be shared. This object is sent to the Agent State Bolt for robot pose sharing.

Each of the three objects spawned by the Global Planner Bolt will be stored as <robot Id,object> Hash Map in the destiny Bolt.

The Dispatcher Bolt will check those Control-Publish Time State object stored in the Bolt instance and every 10ms it will receive a Tuple from the Timer Spout, which will trigger the Dispatcher Bolt to check whether it needs to emit a new Tuple to the Agent State Bolt to start a new controlling/publishing loop.

The Agent State Bolt implements collecting up to date robot information module as shown in Fig. 5. If it gets a Tuple that tells it to calculate a new velocity command, then the Bolt will create a new Agent State object and store all current robot state information and then send it as a Tuple to the next Velocity Compute Bolt. Else if the Tuple asks it to share the robot information to others, the Bolt will fill the Pose Share Message object with the current state information and send it to the Pose Share Publish Bolt for publishing the message to the message broker. After Pose Share Message is published, Agent State Bolt also needs to send back a Tuple to the Dispatcher Bolt to tell it that current job is done new Pose Share task for this robot can be accepted.

The Velocity Compute Bolt contains all other module for velocity command calculation. After a new Agent State object is received, this Bolt will select the correct Agent object from the Hash Map that stores Agent objects from Global Planner Bolt and then execute step 2 to step 6 in Fig. 5. If the calculated velocity command is valid, it will be sent to Velocity Command Publish Bolt to publish the command back to the cloud driver through message broker. Just like Agent State Bolt, this Bolt has to send a Tuple back to Dispatcher Bolt to tell it that the Topology is ready for next calculation for this robot.

As mentioned before, some of the Bolts may cache some runtime information about a robot, however each Bolt can run multiple instances in parallel and how to make sure the Tuple to be sent to the Bolt instance that caches the right robot information is very important to the process logic. In Apache Storm, the organization of connections between instances of different connected Bolts is called Grouping. Here, each Tuple except the one emitted from Timer Spout is attached with a robot id field and Field Grouping based on the id field is used.

The Topology shown in Fig. 6B can be used to control multiple robots and only requires them to send the required information and parameters to the Topology. From the Cloud Computing perspective, the IoT Cloud platform that runs this application Topology can provide robot collision avoidance control services. And with the ability to scale the platform, the application or even a single Bolt in the Topology, such robot control framework can be used in Swarm Robotics that need to control large number of robots and at the same time keep the detailed and complicated control logic and robot model.

**V. EXPERIMENT AND RESULTS**

To verify the application that is developed in this paper, several experiments and tests are carried out.

As the algorithm implemented in this paper has already been tested in the real world multi-robot collision avoidance. This paper will only use simulator to test the application. First,Turtlebot Simulator based on Gazebo and ROS is used to test the validity for real world application. Gazebo[[23](#_ENREF_23)] is a 3D robot simulator which provides support for ROS integration. The Turtlebot Simulator package in ROS uses Gazebo for robot modeling and ROS for controlling and communicating. System deployment and configuration is shown in Fig. 7 and Table 1. Test results are shown in Fig. 8. Form Fig. 8 it can be seen that each Turtlebot can avoid other robots in a very smooth and efficient way.

Table 1 Hardware configuration of the system

|  |  |  |
| --- | --- | --- |
|  | VMs in Cloud | Local host |
| CPU Model | Intel Core i7 9xx | Intel Core i7-2620M |
| CPU Frequency/Mhz | 2933.436 | 2701 |
| Cores | 4 | 2 |
| Thread per core | 1 | 2 |
| Memory/MB | 8192 | 7900 |
| OS | Ubuntu 12.04.4(Linux 3.2.0) | Ubuntu 12.04.5(Linux 3.13.0) |
| Hypervisor | KVM | None |

Fig. 7 Deployment of the application

Fig. 8 Test the application using Turtlebot Simulator

Next, scalability test of the application for more robots is carried out. Since Gazebo models the robot in a very detailed way, it requires lots of computing resources and it is not suitable for simulating large number of robots. Another light weighted robot simulation software called Simbad[[24](#_ENREF_24)] is used in this test. As there is no SLAM(Simultaneous Localization and Mapping)[[25](#_ENREF_25)] module in Simbad and localization pose array cannot be generated, here Gaussian noise is added to the robot pose to create a fake localization pose array for the test.

The most computation-intensive component is the Velocity Compute Bolt, so velocity command delays for different number of robots with different parallelism hint for Velocity Compute Bolt is measured. As shown in Table 1 there are 5 computation nodes with 20 cores in the cluster, to make sure each Bolt instance runs in parallel, the maximum parallelism for Velocity Compute Bolt is limited to 5, while for Agent State Bolt it is set to 2 to see whether increase parallelism of Agent State Bolt can bring better performance. Other components in the application Topology have only one instance for each. And also to make sure the computation load evenly distributed between the instances, the Filed Grouping strategy is changed to a custom module grouping and an index getting value sequentially from 1 to maximum number of robots is assigned to each robot. This index is attached to each message and the module grouping uses result of index mod instance number to select which instance or task the message is sent to.

First NPC(Number of parallelism for Velocity Compute Bolt) is set to 5 and NPS(Number of parallelism for Agent State Bolt) changes from 1 to 2 and test the delays and collision times for NR(Number of robots) range from 5 to 50 to see how many robots the system can serve. Both the control frequency and robot pose share frequency are set to 20 Hertz, which means velocity command latency should be around 50 millisecond for the robots to avoid collide with each other effectively. All robots are arranged on a circle with a radius of 6 meter and centered on the origin of the coordinate and they will go through the center to the antipodal position, then turn around and repeat the process. Each test runs for 300 second. The result is shown in Fig. 9.

Fig. 9 Test result for NPS=2 and NPC=5

Fig. 9 indicates that when the number of robots increases to 25, collision will happen and the average velocity command delay increases to around 57 millisecond. So for NPC less than 5 the maximum number of robots is set to 30. The test results are shown in Fig. 10.

From Fig. 10, it can be seen that when the delay increases to around 60 millisecond collision will happen. However in dense scenarios collision may still happen if the delay is less but near to 60 millisecond. Also increasing the parallelism of Agent State Bolt does not improve the performance, and this is caused by the message transmitting overhead. Both Fig. 9 and Fig. 10 show that with the increase of parallelism for Velocity Compute Bolt, delay of the calculation for new command decreases drastically, which proves that IoT Cloud can be used for real time robot control. Moreover IoT Cloud based application can maintain good performance when the number of robots increases by simply scaling the computation resources in the Cloud. Such scaling ability with real time controlling provide a novel approach for Swarm Robotics or even Swarm Intelligence that contains more than just robotic area. Fig. 11 shows some snapshots of the simulation.

Fig. 10 Test results for different combinationof NPC and NPS

Fig. 11 Snapshots of the test

**VI. CONCLUSION**

This paper provides a novel IoT Cloud based computation framework for Swarm robotics. To demonstrate the viability of the framework for real time control of large number of robots, local collision avoidance algorithm is implemented as an IoT Cloud application. Unlike other research work that tries to minimize computation cost by ignoring important real world factors, this paper adopts complex algorithm that can reflect more details about the real world scenario. However such computation intensive tasks are transferred to the Cloud, so that robots or other intelligent entities can be simplified in both hardware and software. By offloading computation to the IoT Cloud, more complex entities can be studied in Swarm Robotics/Intelligence without trimming off the details of the entity. Such precise implementation of the entity model can lead to deeper insight into its swarm characteristics.

By implementing and testing the collision avoidance algorithm on the IoT Cloud platform, scaling and real time controlling ability of the system is verified. The result shows that by simply scaling the computation resources, one IoT Cloud application can provide service for more robots. Such features will greatly facilitate extending of the population in Swarm robotics and also provide support for large scale Swarm systems.

**ACKNOWLEDGEMENT**

The work carried out in this paper is partially funded by the CSC(Chinese Scholarship Council) of Chinese government.

**REFERENCES**

[1] G. C. Fox，S. Kamburugamuve，R. D. Hartman. Architecture and measured characteristics of a cloud based internet of things[C]. Collaboration Technologies and Systems (CTS), 2012 International Conference on, 2012: 6-12.

[2] Gerardo Beni, "From swarm intelligence to swarm robotics," in *Swarm Robotics*, 1 ed: Springer, 2005, pp. 1-9.

[3] H Woern，Marc Szymanski，Joerg Seyfried. The I-SWARM project[C]. Robot and Human Interactive Communication, 2006. ROMAN 2006. The 15th IEEE International Symposium on, 2006: 492-496.

[4] Marco Dorigo，Elio Tuci，Roderich Groß，et al., "The swarm-bots project," in *Swarm Robotics*, 1 ed: Springer, 2005, pp. 31-44.

[5] Erol Şahin, "Swarm robotics: From sources of inspiration to domains of application," in *Swarm robotics*, 1 ed: Springer, 2005, pp. 10-20.

[6] Gajamohan Mohanarajah，Dominique Hunziker，Raffaello D'Andrea，et al. Rapyuta: A cloud robotics platform[J]. 2014.

[7] D Lorencik，P Sincak. Cloud Robotics: Current trends and possible use as a service[C]. Applied Machine Intelligence and Informatics (SAMI), 2013 IEEE 11th International Symposium on, 2013: 85-88.

[8] Ben Kehoe，Sachin Patil，Pieter Abbeel，et al. A survey of research on cloud robotics and automation[J]. 2015.

[9] Koji Kamei，Shuichi Nishio，Norihiro Hagita，et al. Cloud networked robotics[J]. Network, IEEE, 2012, 26(3): 28-34.

[10] Lucio Agostinho，Leonardo Olivi，Guilherme Feliciano，et al. A cloud computing environment for supporting networked robotics applications[C]. Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on, 2011: 1110-1116.

[11] Jur Van Den Berg，Stephen J Guy，Ming Lin，et al., "Reciprocal n-body collision avoidance," in *Robotics research*, 1 ed: Springer, 2011, pp. 3-19.

[12] Paolo Fiorini，Zvi Shiller. Motion planning in dynamic environments using velocity obstacles[J]. The International Journal of Robotics Research, 1998, 17(7): 760-772.

[13] Jur Van den Berg，Ming Lin，Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation[C]. Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on, 2008: 1928-1935.

[14] Jamie Snape，Jur van den Berg，Stephen J Guy，et al. Independent navigation of multiple mobile robots with hybrid reciprocal velocity obstacles[C]. Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on, 2009: 5917-5922.

[15] Javier Alonso-Mora，Andreas Breitenmoser，Martin Rufli，et al. Optimal reciprocal collision avoidance for multiple non-holonomic robots[M]. Springer, 2013. 2013.

[16] Daniel Hennes，Daniel Claes，Wim Meeussen，et al. Multi-robot collision avoidance with localization uncertainty[C]. Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1, 2012: 147-154.

[17] Daniel Claes，Daniel Hennes，Karl Tuyls，et al. Collision avoidance under bounded localization uncertainty[C]. Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, 2012: 1192-1198.

[18] Stephen J Guy，Jatin Chhugani，Changkyu Kim，et al. Clearpath: highly parallel collision avoidance for multi-agent simulation[C]. Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 2009: 177-187.

[19] L Turnbull，B Samanta. Cloud robotics: Formation control of a multi robot system utilizing cloud infrastructure[C]. Southeastcon, 2013 Proceedings of IEEE, 2013: 1-4.

[20] M Tim Jones. Process real-time big data with Twitter Storm[J]. IBM Technical Library, 2013.

[21] https://zookeeper.apache.org/[J].

[22] http://www.ros.org/[J].

[23] http://gazebosim.org/[J].

[24] Louis Hugues，Nicolas Bredeche, "Simbad: an autonomous robot simulation package for education and research," in *From Animals to Animats 9*, 1 ed: Springer, 2006, pp. 831-842.

[25] Sebastian Thrun，John J Leonard, "Simultaneous localization and mapping," in *Springer handbook of robotics*, 1 ed: Springer, 2008, pp. 871-889.